Towards an Automatic Proof of the Bakery Algorithm

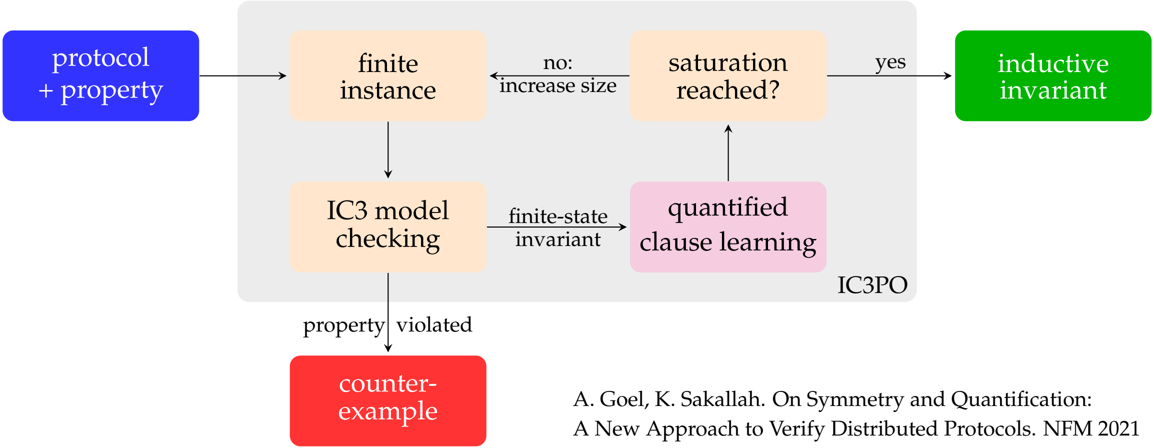
Aman Goel — Amazon Web Services
Stephan Merz — Inria Nancy & LORIA
Karem Sakallah — University of Michigan

WG 2.3 meeting
Trento, October 2023

Overview

- Verification of parameterized algorithms is undecidable
  - verify small instances or use interactive theorem proving
- Automatic synthesis of inductive invariants
  - inductive invariants serve as certificates of correctness
  - IC3PO: learn invariants from finite instances
- Can this be successfully applied in practice?
  - benchmark problem: Bakery mutual exclusion algorithm (Lamport 1974)

IC3PO in a Nutshell



A. Goel, K. Sakallah. On Symmetry and Quantification:
A New Approach to Verify Distributed Protocols. NFM 2021

Exploit structural regularity for generalizing invariants from finite instances

From Ground Instances to Quantified Formulas

1. Quantifier synthesis for symmetric domains

   - assume the following clauses appear for all distinct $i, j, k \in Proc$

     $C(i) \Rightarrow \neg C(j)$

From Ground Instances to Quantified Formulas

1. Quantifier synthesis for symmetric domains
   - assume the following clauses appear for all distinct $i, j, k \in Proc$

$$C(i) \Rightarrow \neg C(j) \qquad \rightsquigarrow \qquad \forall p, q \in Proc : C(p) \land p \neq q \Rightarrow \neg C(q)$$

From Ground Instances to Quantified Formulas

1. Quantifier synthesis for symmetric domains

   - assume the following clauses appear for all distinct $i, j, k \in Proc$

     $C(i) \Rightarrow \neg C(j) \quad \leadsto \quad \forall p, q \in Proc : C(p) \wedge p \neq q \Rightarrow \neg C(q)$

     $A(i) \Rightarrow B(j) \vee B(k)$

From Ground Instances to Quantified Formulas

1. Quantifier synthesis for symmetric domains
   - assume the following clauses appear for all distinct $i, j, k \in Proc$

$$C(i) \Rightarrow \neg C(j) \quad \rightsquigarrow \quad \forall p, q \in Proc : C(p) \land p \neq q \Rightarrow \neg C(q)$$

$$A(i) \Rightarrow B(j) \lor B(k) \quad \rightsquigarrow \quad \forall p \in Proc : A(p) \Rightarrow \exists q \in Proc : p \neq q \land B(q)$$

From Ground Instances to Quantified Formulas

1. Quantifier synthesis for symmetric domains

   - assume the following clauses appear for all distinct $i, j, k \in Proc$

$$C(i) \Rightarrow \neg C(j) \quad \leadsto \quad \forall p, q \in Proc : C(p) \land p \neq q \Rightarrow \neg C(q)$$
$$A(i) \Rightarrow B(j) \lor B(k) \quad \leadsto \quad \forall p \in Proc : A(p) \Rightarrow \exists q \in Proc : p \neq q \land B(q)$$

2. Quantifier synthesis for totally ordered domains

   - take into account order relation (finite instance $N = 3$)

$$P(1) \Rightarrow Q(2) \land Q(3)$$
$$P(2) \Rightarrow Q(3)$$

From Ground Instances to Quantified Formulas

1. Quantifier synthesis for symmetric domains

   - assume the following clauses appear for all distinct $i, j, k \in Proc$

$$C(i) \Rightarrow \neg C(j) \quad \rightsquigarrow \quad \forall p, q \in Proc : C(p) \land p \neq q \Rightarrow \neg C(q)$$

$$A(i) \Rightarrow B(j) \lor B(k) \quad \rightsquigarrow \quad \forall p \in Proc : A(p) \Rightarrow \exists q \in Proc : p \neq q \land B(q)$$

2. Quantifier synthesis for totally ordered domains

   - take into account order relation (finite instance $N = 3$)

$$\begin{array}{l} P(1) \Rightarrow Q(2) \land Q(3) \\ P(2) \Rightarrow Q(3) \end{array} \quad \rightsquigarrow \quad \forall i, j \in 0 \,..\, N : P(i) \land 0 < i < j \Rightarrow Q(j)$$

Quantifiers formally express symmetries in properties

The Bakery Algorithm

variables $num = [i \in P \mapsto 0], flag = [i \in P \mapsto \textbf{false}]$
process $self \in P$:
  variables $unread = \{\}, max = 0$;
p1: **while true**:
  $unread := P \setminus \{self\}; max := 0; flag[self] := \textbf{true}$;
p2: **for** $nxt \in unread$:
  if $num[nxt] > max$: $max := num[nxt]$;
  $unread := unread \setminus \{nxt\}$;
p3: $num[self] :> max$;
p4: $flag[self] := \textbf{false}; unread := P \setminus \{self\}$;
p5: **for** $nxt \in unread$:
  await $\neg flag[nxt]$;
p6: **await** $(num[nxt] = 0) \lor self \ll nxt$;
  $unread := unread \setminus \{nxt\}$
cs: **skip**;
p7: $num[self] := 0$

$num[i]$: ticket number of process i
$flag[i]$: process i draws a ticket

iterate over processes to determine highest ticket number currently in use

pick some higher ticket number

iterate over processes:
– make sure the process doesn't draw a ticket
– wait for the process to have lower priority

signal exit by giving up ticket

$P \triangleq 1\,..\,N \qquad i \ll j \triangleq num[i] < num[j] \lor (num[i] = num[j] \land i \leq j)$

Formal Specifications of the Bakery Algorithm

- Landmark algorithm for ensuring mutual exclusion
  - intuition: organize a queue where customers draw tickets
- Effect of concurrent reads and writes
  1. atomic reads and writes: memory operations never interfere
  2. safe registers: a read overlapping a write returns an arbitrary (type-correct) value
- Existing TLA$^+$ specifications and hand-written proofs
  - we will discuss the non-atomic version, but the results apply to both

Applying IC3PO to the Bakery

1. Encode existing TLA$^+$ specification in Ivy
   - typed, relational input language, e.g., represent $i \in \mathit{unread}[j]$ by $\mathit{unread}(i,j)$
2. Run IC3PO model checker for proving mutual exclusion
   - initial domain size: 3 processes, 3 ticket numbers
   - saturation at 4 processes, 3 ticket numbers
   - 42 quantified invariants generated
3. Rewrite IC3PO invariant as a TLA$^+$ formula
   - group similar clauses for different control points, reorient implications
   - use TLAPS to check that the TLA$^+$ version of the invariant is inductive

Two Invariants for the Non-Atomic Bakery

$HInv \triangleq TypeOK \land \forall i \in P : HIInv(i)$

$HIInv(i) \triangleq$

A1 $\land\ pc[i] \in \{\text{“p1”}, \text{“p2”}\} \Rightarrow num[i] = 0$

A2 $\land\ num[i] = 0 \Rightarrow pc[i] \in \{\text{“p1”}, \text{“p2”}, \text{“p3”}, \text{“p7”}\}$

B1 $\land\ pc[i] \in \{\text{“p2”}, \text{“p3”}\} \Rightarrow flag[i]$

B2 $\land\ flag[i] \Rightarrow pc[i] \in \{\text{“p1”}, \text{“p2”}, \text{“p3”}, \text{“p4”}\}$

C $\land\ pc[i] \in \{\text{“p5”}, \text{“p6”}\}$
$\quad \Rightarrow \forall j \in (P \setminus unread[i]) \setminus \{i\} : After(j, i)$

D $\land\ \land\ pc[i] = \text{“p6”}$
$\quad \land\ \lor\ pc[nxt[i]] = \text{“p2”} \land i \notin unread[nxt[i]]$
$\quad\quad \lor\ pc[nxt[i]] = \text{“p3”}$
$\quad \Rightarrow max[nxt[i]] \geq num[i]$

E $\land\ pc[i] = \text{“cs”} \Rightarrow \forall j \in P \setminus \{i\} : After(j, i)$

$After(j, i) \triangleq$
$\land\ num[i] > 0$
$\land\ \lor\ pc[j] = \text{“p1”}$
$\quad \lor\ pc[j] = \text{“p2”} \land (i \in unread[j] \lor max[j] \geq num[i])$
$\quad \lor\ pc[j] = \text{“p3”} \land max[j] \geq num[i]$
$\quad \lor\ \land\ pc[j] \in \{\text{“p4”}, \text{“p5”}, \text{“p6”}\} \land i \ll j$
$\quad\quad \land\ pc[j] \in \{\text{“p5”}, \text{“p6”}\} \Rightarrow i \in unread[j]$
$\quad \lor\ pc[j] = \text{“p7”}$

$MInv \triangleq \forall i \in P : MIInv(i)$

$MIInv(i) \triangleq$

a $\land\ pc[i] \in \{\text{“p4”}, \text{“p5”}, \text{“p6”}, \text{“cs”}\} \Rightarrow num[i] \neq 0$

b1 $\land\ pc[i] \in \{\text{“p2”}, \text{“p3”}\} \Rightarrow flag[i]$

b2 $\land\ pc[i] \in \{\text{“p5”}, \text{“p6”}\} \land flag[i] \Rightarrow \forall j \in P \setminus \{i\} :$
$\quad \land\ pc[j] \in \{\text{“p5”}, \text{“p6”}\} \Rightarrow i \in unread[j]$
$\quad \land\ pc[j] = \text{“p6”} \Rightarrow i \neq nxt[j]$
$\quad \land\ pc[j] = \text{“cs”} \Rightarrow i = nxt[j] \lor j = nxt[j]$

c $\land\ pc[i] \in \{\text{“p5”}, \text{“p6”}\} \Rightarrow \forall j \in P \setminus unread[i] :$
$\quad \land\ pc[j] = \text{“p2”} \Rightarrow i \in unread[j] \lor max[j] \geq num[i]$
$\quad \land\ pc[j] = \text{“p3”} \Rightarrow max[j] \geq num[i]$
$\quad \land\ pc[j] \in \{\text{“p4”}, \text{“p5”}, \text{“p6”}\} \Rightarrow i \ll j$

d1 $\land\ pc[i] = \text{“p6”} \land pc[nxt[i]] = \text{“p2”}$
$\quad \Rightarrow i \in unread[nxt[i]] \lor max[nxt[i]] \geq num[i]$

d2 $\land\ pc[i] = \text{“p6”} \land pc[nxt[i]] = \text{“p3”} \land flag[nxt[i]]$
$\quad \Rightarrow max[nxt[i]] \geq num[i]$

e $\land\ pc[i] = \text{“cs”} \Rightarrow \forall j \in P \setminus \{i\} :$
$\quad \land\ pc[j] = \text{“p2”} \Rightarrow i \in unread[j] \lor max[j] \geq num[i]$
$\quad \land\ pc[j] = \text{“p3”} \Rightarrow max[j] \geq num[i]$
$\quad \land\ pc[j] = \text{“p4”} \Rightarrow i \ll j$
$\quad \land\ pc[j] \in \{\text{“p5”}, \text{“p6”}\} \Rightarrow i \ll j \land i \in unread[j]$
$\quad \land\ pc[j] \neq \text{“cs”}$

Comparing the Two Invariants

- The two invariants are structurally similar
  - based on the same atomic propositions
  - superficial syntactic differences due to generation from CNF formulas
- *HInv* uses auxiliary predicate $\mathit{After}(j, i)$
  - the implications C and E (resp., c and e) assert similar conditions
  - auxiliary predicate abstracts this similarity

A Closer Look at Parts B / b

B1 $\land\, pc[i] \in \{\text{"p2"}, \text{"p3"}\} \Rightarrow flag[i]$

B2 $\land\, flag[i] \Rightarrow pc[i] \in \{\text{"p1"}, \text{"p2"}, \text{"p3"}, \text{"p4"}\}$

b1 $\land\, pc[i] \in \{\text{"p2"}, \text{"p3"}\} \Rightarrow flag[i]$

b2
$$\left\{\begin{array}{l} \land\, flag[i] \land pc[i] \in \{\text{"p5"}, \text{"p6"}\} \Rightarrow \forall j \in P \setminus \{i\} : \\ \quad \land\, pc[j] \in \{\text{"p5"}, \text{"p6"}\} \Rightarrow i \in unread[j] \\ \quad \land\, pc[j] = \text{"p6"} \Rightarrow i \neq nxt[j] \\ \quad \land\, pc[j] = \text{"cs"} \Rightarrow i = nxt[j] \lor j = nxt[j] \end{array}\right.$$

- Assertions about the flag being set
  - B1 and b1 are identical
  - B2 implies b2
- The computer-generated invariant is weaker
  - inspecting the code shows that the flag cannot be set beyond p4
  - IC3PO propagates predicates using backward reachability analysis

Summary

- IC3PO successfully generated an inductive invariant for Bakery
  - based on existing specifications, faithfully rewritten in Ivy
  - inductive invariants serve as certificates of correctness
- The synthesized invariant is remarkably similar to a human-written one
  - both capture the relevant arguments for proving mutual exclusion
  - machine-generated invariant is a little more permissive
- Perspectives
  - directly handle interesting fragment of TLA^+: avoid manual encoding in Ivy
  - handle more case studies and assess scalability